Generating Sequential Space-filling Designs Using Genetic Algorithms and Monte Carlo Methods

Karel Crombecq[1] and Tom Dhaene[2]

[1] University of Antwerp, 2020 Antwerp, Belgium
Karel.Crombecq@ua.ac.be
[2] Ghent University, 9050 Ghendt, Belgium

Abstract. Accurate, high fidelity computer simulations are often used instead of real-life experiments, in order to reduce the overall time, cost and/or risk. Because these simulations can be very time-expensive, global surrogate modelling is regularly employed to improve efficiency. After performing simulations at different locations in the design space, a global surrogate model is trained that mimics the original simulator, but can be evaluated much faster. In sequential design or active learning, the simulations are performed one by one, and global surrogate models are built after each iteration, in order to avoid over- or undersampling. In this paper, the authors compare a Monte Carlo method and an optimization-based approach using genetic algorithms for sequentially generating space-filling experimental designs. It is shown that Monte Carlo methods perform better than genetic algorithms for this specific problem.

Keywords: surrogate modelling, active learning, sequential design, Monte Carlo, genetic algorithm, space-filling

1 Introduction

For many modern engineering problems, accurate high fidelity simulations are often used instead of controlled real-life experiments, in order to reduce the overall time, cost and/or risk. These simulations are used by the engineer to understand and interpret the behaviour of the system under study and to identify interesting regions in the design space. They are also used to understand the relationships between the different input parameters and how they affect the outputs.

However, the simulation of one single instance of a complex system with multiple inputs (also called factors or variables) and outputs (also called responses) can be a very time-consuming process. For example, Ford Motor Company reported on a crash simulation for a full passenger car that takes 36 to 160 hours to compute [3]. Because of this long computational time, using this simulation directly is still impractical for engineers who want to explore, optimize or gain insight into the system.

In our approach, we assume the system under study is a black box, with little or no additional information available about its inner working except for the output it generates. This means that, without running simulations, nothing is known about the behaviour of the function, and no assumptions can be made about continuity or linearity or any other mathematical properties the system might have. A final assumption is that the simulator is deterministic, meaning that the same output is produced if the simulator is run twice with the same input values. This is not the same as saying that there is a complete absence of noise; indeed, noise may be introduced by the way the simulator models and discretises the real world. It only implies that, even if there is noise in the simulation outputs, the noise will be identical for two simulation runs with the same inputs.

The goal of *global* surrogate modelling is to find a function that mimics the original system, but can be computed much faster. This function is constructed by performing multiple simulations (called samples) at key points in the design space, analyzing the results, and selecting a model that approximates the samples and the overall system behavior quite well.

Please note that *global* surrogate modelling differs from *local* surrogate modelling in the way the surrogate models are employed. In *local* surrogate modelling, local models are used to guide the optimization algorithm towards a global optimum. The local models are discarded afterwards. In *global* surrogate modelling, the goal is to create a model that approximates the behaviour of the simulator on the entire domain, so that the surrogate model can then be used as a full replacement for the original simulator, or can be used to explore the design space. Thus, the goal of global surrogate modelling is to overcome the long computational time of the simulator by providing a fast but accurate approximation, based on a one-time upfront modelling effort. In this paper, we are only concerned with global surrogate modelling.

It is clear that the choice of the data points (or samples) is of paramount importance to the success of the surrogate modelling task. Intuitively, the data points must be spread out in such a way as to convey a maximum amount of information about the behaviour of simulator. This is a non-trivial challenge, since little or nothing is known about this (black-box) simulator in advance. From now on, we will refer to the entire set of samples that were selected for evaluation as the experimental design.

In traditional design of experiments (DOE), the experimental design is chosen based only on information that is available before the first simulation. This experimental design is then fed to the simulator, which evaluates all the selected data points. Finally, a surrogate model is built using this data. This is essentially a one-shot approach, as all the data points are chosen at once and the modelling algorithm proceeds from there, without evaluating any additional samples later on. Examples of popular one-shot space-filling design are fractional designs [12], Latin hypercubes [11] and orthogonal arrays [2].

Sequential design (which is also known as adaptive sampling [8] or active learning [13]) further improves on this approach by transforming the one-shot

algorithm into an iterative process. Sequential design methods generate samples one by one, allowing for a more integrated approach with the surrogate modelling task. After each simulation, new models are built, and the accuracy of these models is estimated. If the target accuracy is reached, the algorithm is halted. Otherwise, another sample is selected and the process starts all over again.

Sequential design methods have several advantages over the classical one-shot approach. When the one-shot approach is used, too many samples may be evaluated to achieve the desired accuracy (oversampling) or too little samples may have been evaluated (undersampling), in which case one must completely restart the experiment or resolve to sequential methods to improve the original design. These two issues are avoided by sequential methods. Additionally, sequential methods do not need to know the total number of samples in advance. In the one-shot approach, one must guess the total number of samples that will be needed to achieve the desired accuracy, in order to pick the right experimental design. In the context of a black-box simulator, where nothing is known about the simulator in advance, this is a huge disadvantage, because there is no information at all available to make an educated guess on the number of samples that will be needed.

In this paper, we study the sequential generation of space-filling designs. Space-filling designs attempt to spread out the samples as evenly as possible, in order to get as much information about the entire design space as possible. We will investigate why it is difficult to sequentially generate good space-filling designs, and we will compare two approaches to tackling this problem: optimization using genetic algorithms, and Monte Carlo methods.

2 Space-filling Experimental Design Criteria

From now on, we will consider the d-dimensional experimental design $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ containing n samples $\mathbf{p}_i = \left(p_i^1, p_i^2, \ldots, p_i^d\right)$ in the (hyper)cube $[-1, 1]^d$. This experimental design P will be constructed by selecting samples one by one, without knowing the total number of samples n at any point during the construction process. In order to evaluate the space-filling qualities of the final design, we consider two criteria.

2.1 Intersite Distance

First and foremost, the generated design should be space-filling. Intuitively, a space-filling design is an experimental design in which the points are spread out evenly over the design space. However, there are several ways do define this property mathematically. Over the course of the years, many different space-filling criteria have been proposed. Depending on the criterion, the optimal design P will look differently.

A popular and intuitive choice is the maximin criterion or *intersite distance*, which is defined as follows:

$$\text{idist}(P) = \min_{\mathbf{p_i},\mathbf{p_j}\in P} \sqrt{\sum_{k=1}^{d} \left\| p_i^k - p_j^k \right\|^2} \tag{1}$$

The intersite distance of an experimental design P is the smallest distance between two points in the design. By maximizing this criterion, one obtains a design in which all points are as far away from each other as possible. This criterion has been used by many authors before [1, 6, 7, 9, 10, 15]. Because of its intuitive appeal and all-round popularity, this criterion will be used in this paper as well.

2.2 Projective Distance

Secondly, a good space-filling design should also have good projective properties. This is also called the non-collapsing property by some authors [1]. An experimental design P has good projective properties if, for every point $\mathbf{p}_i$, each value p_i^j is strictly unique, and as different from the other values as possible. This property also means that, when the experimental design is projected from d-dimensional space to $(d-1)$-dimensional space along one of the axes, no two points are ever projected onto the same location.

The quality of a design in terms of its projective properties can be defined as the minimum *projected distance* of points from each other:

$$\begin{aligned} \text{pdist}(P) &= min_{\mathbf{p_i},\mathbf{p_j}\in P} \min_{1\le k\le d} \left| p_i^k - p_j^k \right| \\ &= min_{\mathbf{p_i},\mathbf{p_j}\in P} \left\| \mathbf{p_i} - \mathbf{p_j} \right\|_{-\infty} \end{aligned} \tag{2}$$

where $\|x\|_{-\infty}$ is the minus infinity norm. This is an important property if it is unknown in advance if there are design parameters included in the experiment which have little or no influence on the response. If this is the case, two samples which differ only for this design parameter can be considered the same point, and evaluating the same point twice is a waste of computational time. Therefore, each sample should have unique values for each design parameter. Preferably, when all the points are projected onto one of the axis, the resulting design should be space-filling as well. Ideally, all the projected points should be equidistant.

An experimental design with optimal (equidistant after projection) non-collapsing points will not suffer a performance hit when one of the design parameters turns out to be irrelevant. This is very important in the context of a black-box simulator, since it may not be known in advance how relevant and influential each parameter is to the response.

3 Experimental Setup

Generating an experimental design that satisfies the two criteria mentioned in the previous section is a multi-objective optimization problem. Starting with

two initial points (for example, opposing corner points), a new point is selected by finding a location in the design space that maximizes both the intersite and projective distance. Many different methods have been proposed to solve such multi-objective optimization problems efficiently. The simplest approach is to combine the different objectives in a single aggregate objective function. This solution is only acceptable if the scale of both objectives is known, so that they can be combined into a formula that gives each objective equal weight. Fortunately, in the case of the intersite and projective distance, this is indeed the case.

The aggregate intersite and projective distance criterion is defined as follows:

$$\text{dist}(P) = \frac{(n+1)^{\frac{1}{d}-1}}{2}\text{idist}(P) + \frac{n+1}{2}\text{pdist}(P) \tag{3}$$

However, using this function as the objective is not yet ideal. Consider a design, for which two points already have an intersite distance of 0.1. Then all new candidates that lie further away from the other points than 0.1 result in the same objective score, since the minimum intersite distance does not change. However, it is preferable to choose the point farthest away from the existing points. Therefore, instead of computing the distance of all points from each other, we just compute the distance of the new point from previous points, and optimize this function.

The final objective function, which scores a new candidate point $\mathbf{p}$ when it is added to an existing design P, is defined as:

$$\text{dist}(P, \mathbf{p}) = \frac{(n+1)^{\frac{1}{d}-1}}{2}\min_{\mathbf{p_i}\in P}\sqrt{\sum_{k=1}^{d}\left|p_i^k - p^k\right|^2} + \frac{n+1}{2}\min_{\mathbf{p_i}\in P}\left\|\mathbf{p_i} - \mathbf{p}\right\|_{-\infty} \tag{4}$$

In this paper, we will compare two approaches to finding the best location for the next point at each iteration. The first one is a Monte Carlo method. In this method a large number of uniformly distributed random points is generated, and for each candidate $\mathbf{p}$, the objective function $\text{dist}(P, \mathbf{p})$ is calculated and the best candidate is selected as the new point to be added to P. In the second method, the genetic algorithm toolbox from Matlab will be used to optimize the objective function and find the best next candidate. Both methods will be compared for different settings, and the final designs be evaluated on the `idist` and `pdist` criteria to compare both approaches.

At each iteration, the Monte Carlo method will generate kn random points, where n is the number of samples evaluated thus far, and k is an algorithm parameter. It is expected that, for larger k, the quality of the design will improve. In this study, the following values for k were considered: $50, 250, 2000, 10000, 50000$.

For the genetic algorithm, the implementation from the Matlab Genetic Algorithm and Direct Search Toolbox (version 3.0) was used. Most of the options were kept at their default values, but some were changed in order to improve the performance. The default mutation function (which offsets each input by

a value drawn from a gaussuan distribution) wasn't usable, because it did not respect the boundary constraints (each input must lie in $[-1, 1]$). It was changed to a mutation function that changes each input with a chance of 0.01 to a random values in the $[-1, 1]$ interval. Preliminary results have shown that playing with the crossover/mutation fraction settings, changing the elite behaviour etc does not affect the outcome much, so these settings were kept at their default values. This experiment was repeated for different numbers of generations: $50, 100, 250, 1000, 2000$.

All these experiments were carried out using the SUMO Toolbox research platform [4, 5]. This freely available Matlab toolbox, designed for adaptive surrogate modelling and sampling, has excellent extensibility, making it possible for the user to add, customize and replace any component of the sampling and modelling process. Because of this, SUMO was the ideal choice for conducting this experiment[3]. Because both the Monte Carlo and genetic algorithms use random numbers, each experiment was repeated 10 times to get a good average of the performance of the methods.

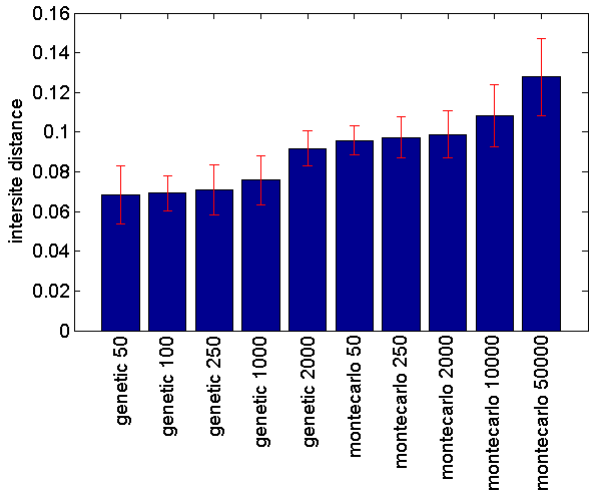
4 Results

The results for the intersite and projective distance values are shown in Fig. 1. These plots show the average intersite and projective distance of each method, after generating 144 points in a $2D$ input space using the algorithms described in the previous section. It is clear that, in both cases, the Monte Carlo method outperforms the genetic algorithm.

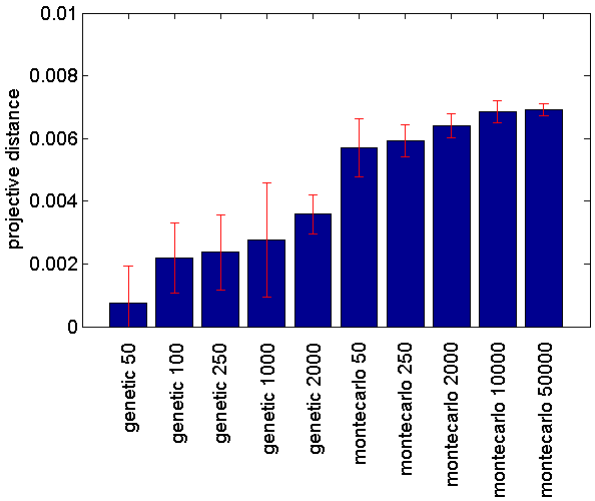
Figure 2 shows the time it took to generate the design for all the methods. Even though the final designs generated by the genetic algorithm are considerably worse than the ones generated by the Monte Carlo method, the genetic algorithm requires much more time to generate them. The genetic algorithm with 2000 generations takes 3 times longer than the Monte Carlo method with $k = 50$, but still produces worse results. It is also noticeable that the difference between 50 generations and 2000 generations is smaller than the difference between $k = 50$ and $k = 50000$, while the difference in elapsed time is larger for the genetic algorithm. This indicates that the rate at which the genetic algorithm improves is actually lower than the improvement rate for the Monte Carlo method. So no matter how many generations are computed, there will always be a Monte Carlo alternative that requires less time to get the same result.

It is clear that genetic algorithms (and optimization methods in general), which are usually considered a better choice than a naive Monte Carlo approach, perform worse in this test case. In order to understand why this is happening, Fig. 3 shows the optimization surfaces of the intersite distance, projective distance and the sum of these two (as defined by (3)), for 12 2D points spread out in a space-filling manner. The intersite distance produces an optimization surface with a considerable number of local optima. But this does not even come close

[3] The SUMO Toolbox v7.0 can be downloaded from `http://www.sumo.intec.ugent.be`.



(a) Intersite distance



(b) Projective distance

Fig. 1. These figures show the average intersite and projective distance of each method, after generating 144 points in a $2D$ input space using the algorithms described in Sect. 3. Each experiment was repeated 10 times, and the standard deviation is shown as well. It is clear that in both cases, the Monte Carlo method performs considerably better than the genetic algorithm.

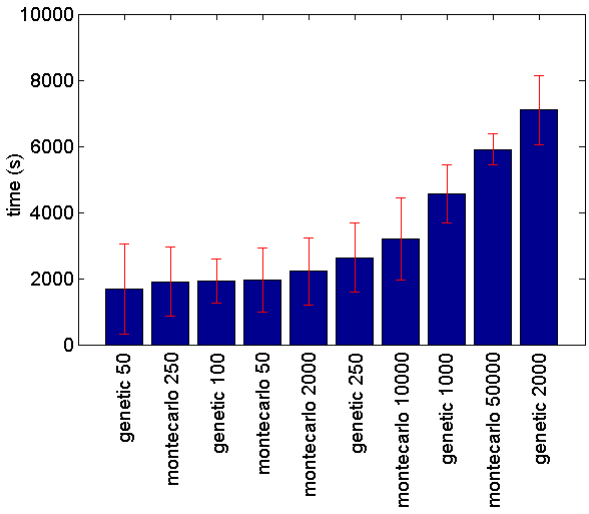


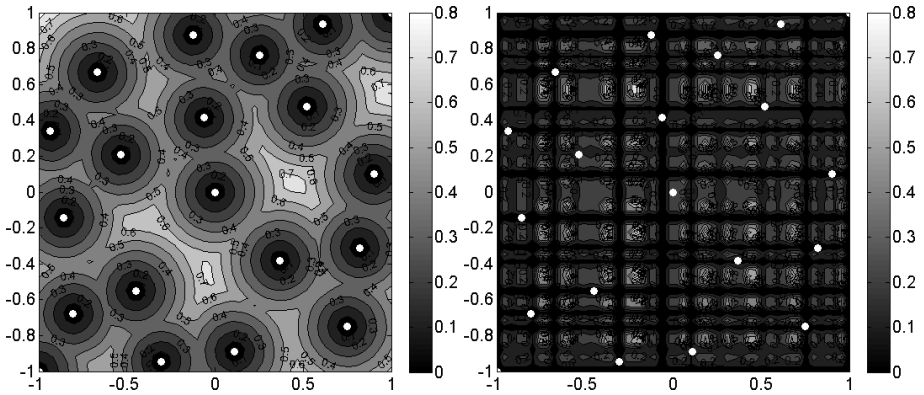
Fig. 2. This figure shows each experiment, sorted by the average time it took to generate a 144-point experimental design. Note that the genetic algorithm requires much more time, while producing worse designs.

to the number of local optima for the projective distance. In fact, the projective distance surface always has $(n+1)^d$ local optima, and only one of them is the global optimum. This optimization surface is so difficult, that it is practically impossible to optimize in an acceptable timeframe. When these two criteria are combined, the resulting optimization surface is even more erratic. This explains why the genetic algorithm quickly gets stuck in a local optimum, and does not manage to get out of it, no matter how many generations are computed.

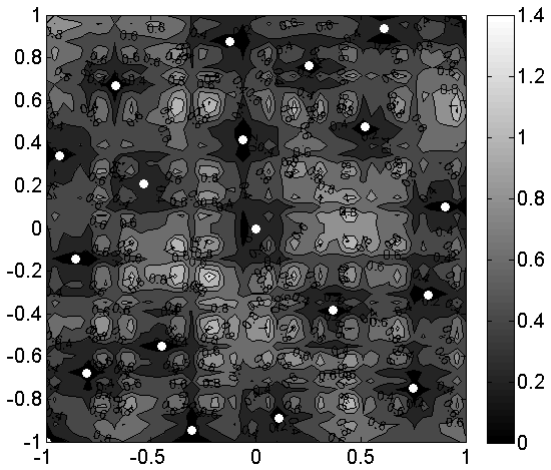
5 Conclusion

This study shows that, when sequentially generating space-filling experimental designs, Monte Carlo methods are prefered above genetic algorithms or other optimization methods. This can be explained by the extremely complex and multimodal optimization surface obtained by adding the intersite and projective distance. It is possible that other optimization methods might perform better than the genetic algorithm implementation from the Matlab toolbox used in this study. However, the authors find it unlikely that any optimization method will do better than the Monte Carlo approach, considering the nature of the optimization surface.

Preliminary experiments have shown that the results published in this paper also hold in higher dimensions, and for different criteria, such as the ϕ_p criterion proposed by [9]. In subsequent publications, these preliminary results will be examined and expanded upon.



(a) Intersite distance (b) Projective distance



(c) Intersite + projective distance

Fig. 3. The optimization surfaces for the intersite and projective distance criteria, as well as the sum of both criteria, for 12 points in a 2D space. Due to the large number of local optima, optimization methods have a lot of trouble finding a globally optimal solution.

The ultimate goal of this experiment is to develop highly efficient sequential space-filling algorithms that can compete with proven and popular one-shot experimental design techniques such as the optimized Latin hypercube [1, 14]. These methods will use Monte Carlo methods as the optimization method of choice, as opposed to global optimization methods. Finally, hybrid methods will also be investigated. Hybrid methods use Monte Carlo to find promising locations, and then perform a local optimization to further improve the initial result.

References

1. van Dam, E.R., Husslage, B., den Hertog, D., Melissen, H.: Maximin latin hypercube design in two dimensions. Operations Research 55(1), 158–169 (2007)
2. Fang, K.T.: Experimental design by uniform distribution. Acta Mathematice Applicatae Sinica 3, 363–372 (1980)
3. Gorissen, D., Crombecq, K., Hendrickx, W., Dhaene, T.: Adaptive distributed metamodeling. High Performance Computing for Computational Science - VECPAR 2006 4395, 579–588 (2007)
4. Gorissen, D., Tommasi, L.D., Crombecq, K., Dhaene, T.: Sequential modeling of a low noise amplifier with neural networks and active learning. Neural Computation & Applications 18(5), 485–494 (2009)
5. Gorissen, D., Turck, F.D., Dhaene, T.: Evolutionary model type selection for global surrogate modeling. Journal of Machine Learning Research 10(1), 2039–2078 (2009)
6. Johnson, M., Moore, L., Ylvisaker, D.: Minimax and maximin distance designs. Journal of Statistical Planning and Inference 26, 131–148 (1990)
7. Joseph, V.R., Hung, Y.: Orthogonal-maximin latin hypercube designs. Statistica Sinica 18, 171–186 (2008)
8. Lehmensiek, R., Meyer, P., Müller, M.: Adaptive sampling applied to multivariate, multiple output rational interpolation models with application to microwave circuits. International Journal of RF and Microwave Computer-Aided Engineering 12(4), 332–340 (2002)
9. Morris, M.D., Mitchell, T.J.: Exploratory designs for computer experiments. Journal of Statistical Planning and Inference 43, 381–402 (1995)
10. Rennen, G., Husslage, B., Dam, E.V., Hertog, D.D.: Nested maximin latin hypercube designs. Tech. rep., Tilburg University (2009)
11. Simpson, T.W., Lin, D.K.J., Chen, W.: Sampling strategies for computer experiments: Design and analysis. International Journal of Reliability and Applications 2(3), 209–240 (2001)
12. Simpson, T.W., Peplinski, J., Koch, P.N., Allen, J.K.: Metamodels for computer-based engineering design: Survey and recommendations. Engineering with Computers 17(2), 129–150 (2001)
13. Sugiyama, M.: Active learning in approximately linear regression based on conditional expectation of generalization error. Journal of Machine Learning Research 7, 141–166 (2006)
14. Viana, F.A.C., Venter, G., Balabanov, V.: An algorithm for fast optimal latin hypercube design of experiments. International Journal for Numerical Methods in Engineering (2009)
15. Ye, K.Q., Li, W., Sidjianto, A.: Algorithmic construction of optimal symmetric latin hypercube designs. Journal of Statistical Planning and Inference 90(1), 145–159 (2000)